(2)

AVF Control Number: NBS87VHFS530_2

Ada Compiler
VALIDATION SUMMARY REPORT:

Certificate Number: 880115S1.09017
Honeywell Bull, Inc.
ONE PLUS Ada Compiler, Version 1.1
Host:
DPS 6 PLUS/420
Target:
DPS 6 PLUS/420

Completion of On-Site Testing:
15 January 1988

Prepared By:
Software Standards Validation Group
Institute for Computer Sciences and Technology
National Bureau of Standards
Building 225, Room A266
Gaithersburg, Maryland 20899

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C. 20301-3081

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETEING FORM
1. REPORT NUMBER 2. GOVT ACCESSION NO. A122 570	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (*and Subtitle*) Ada Compiler Validation Summary Report: Honeywell Bull, Inc., ONE PLUS Ada Compiler, Version 1.1, DPS 6 PLUS/420 (Host and Target).	5. TYPE OF REPORT & PERIOD COVERED 15 Jan 1988 to 15 Jan 1989 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(*s*) National Bureau of Standards, Gaithersburg, Maryland, U.S.A.	8. CONTRACT OR GRANT NUMBER(*s*)
9. PERFORMING ORGANIZATION AND ADDRESS National Bureau of Standards, Gaithersburg, Maryland, U.S.A.	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081	12. REPORT DATE 15 January 1988 13. NUMBER OF PAGES 38 p.
14. MONITORING AGENCY NAME & ADDRESS (*If different from Controlling Office*) National Bureau of Standards, Gaithersburg, Maryland, U.S.A.	15. SECURITY CLASS (*of this report*) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (*of this Report*)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (*of the abstract entered in Block 20. If different from Report*)

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS (*Continue on reverse side if necessary and identify by block number*)

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (*Continue on reverse side if necessary and identify by block number*)

ONE PLUS Ada Compiler, Version 1.1, Honeywell Bull, Inc., National Bureau of Standards, DPS 6 PLUS/420 under HVS 6 PLUS V1.0, ACVC 1.9.

DD FORM 1473   EDITION OF 1 NOV 65 IS OBSOLETE
1 JAN 73   S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

Ada Compiler Validation Summary Report:

Compiler Name: ONE PLUS Ada Compiler, Version 1.1

Certificate Number: 880115S1.09017

Host:
DPS 6 PLUS/420 under HVS 6 PLUS V1.0

Target:
DPS 6 PLUS/420 under HVS 6 PLUS V1.0

Testing Completed 15 January 1988 Using ACVC 1.9

This report has been reviewed and is approved.

[illegible]

Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division
National Bureau of Standards
Gaithersburg, MD 20899

[illegible]

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria, VA 22311

Virginia L. Castor

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC 20301

EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the ONE PLUS Ada Compiler, Version 1.1, using Version 1.9 of the Ada Compiler Validation Capability (ACVC). The ONE PLUS Ada Compiler, Version 1.1 is hosted on a DPS 6 PLUS/420 operating under HVS 6 PLUS V1.0. Programs processed by this compiler may be executed on a DPS 6 PLUS/420 operating under HVS 6 PLUS V1.0.

On-site testing was performed 11 January 1988 through 15 January 1988 at Billerica, MA, under the direction of the Software Standards Validation Group, Institute for Computer Sciences and Technology, National Bureau of Standards (AVF), according to Ada Validation Organization (AVO) policies and procedures. At the time of testing, version 1.9 of the ACVC comprised 3122 tests of which 25 had been withdrawn. Of the remaining tests, 241 were determined to be inapplicable to this implementation. Results for processed Class A, C, D, and E tests were examined for correct execution. Compilation listings for Class B tests were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors. The remaining 2856 tests were passed. The results of validation are summarized in the following table:

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	186	499	549	248	166	98	142	326	135	36	232	3	236	2856
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	18	74	126	0	0	0	1	1	2	0	2	0	17	241
Withdrawn	2	13	2	0	0	1	2	0	0	0	2	1	2	25
TOTAL	206	586	677	248	166	99	145	327	137	36	236	4	255	3122

The AVF concludes that these results demonstrate acceptable conformity to ANSI/MIL-STD-1815A Ada.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report ~~(VSR)~~ describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

This information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent but permitted by the Ada Standard. Six classes of test are used. These tests are designed to perform checks at compile time, at link time, and during execution.

[illegible]

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

> To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
>
> To attempt to identify any unsupported language constructs required by the Ada Standard
>
> To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted Software Standards Validation Group, National Bureau of Standards according to policies and procedures established by the Ada Validation Organization (AVO). On-site testing was conducted from 11 January 1988 through 15 January 1988 at Billerica, MA.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

> Ada Information Clearinghouse
> Ada Joint Program Office
> OUSDRE
> The Pentagon, Rm 3D-139 (Fern Street)
> Washington DC 20301-3081

or from:

> Software Standards Validation Group
> Institute for Computer Sciences and Technology
> National Bureau of Standards
> Building 225, Room A266
> Gaithersburg, Maryland 20899

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983.

2. Ada Compiler Validation Procedures and Guidelines. Ada Joint Program Office, 1 January 1987.

3. Ada Compiler Validation Capability Implementers' Guide. SofTech, Inc., December 1986.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada Commentary	An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. In the context of this report, the AVF is responsible for conducting compiler validations according to established procedures.
AVO	The Ada Validation Organization. In the context of this, report, the AVO is responsible for establishing procedures for compiler validations.

Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Language Maintenance	The Language Maintenance Panel (LMP) is a committee established by the Ada Board to recommend interpretations and Panel possible changes to the ANSI/MIL-STD for Ada.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	An Ada program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn test	An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. However, no checks are performed during execution to see if the test objective had been met. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of these units is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: ONE PLUS Ada Compiler, Version V1.1

ACVC Version: 1.9

Certificate Number:		880115S1.09017
Host Computer:		
	Machine:	DPS 6 PLUS/420
	Operating System:	HVS 6 PLUS V1.0
	Memory Size:	7168K WORDS
Target Computer:		
	Machine:	DPS 6 PLUS/420
	Operating System:	HVS 6 PLUS V1.0
	Memory Size:	7168K WORDS
Communications Network:		NONE

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

- Capacities.

  The compiler correctly processes tests containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See test D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation processes 64 bit integer calculations. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation supports the additional predefined types SHORT_INTEGER, LONG_INTEGER, and LONG_FLOAT in the package STANDARD. (See tests B86001BC and B86001D.)

- Based literals.

  An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR or CONSTRAINT_ERROR during execution. This implementation raises NUMERIC_ERROR during execution. (See test E24101A.)

- Expression evaluation.

  Apparently all default initialization expressions for record components are evaluated before any value is checked to belong to a component's subtype. (See test C32117A.)

Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

This implementation uses no extra bits for extra precision. This implementation uses all extra bits for extra range. (See test C35903A.)

Apparently NUMERIC_ERROR is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

Apparently NUMERIC_ERROR is raised when a literal operand in a fixed point comparison or membership test is outside the range of the base type. (See test C45252A.)

Apparently underflow is not gradual. (See tests C45524A..Z.)

- Rounding.

The method used for rounding to integer is apparently round away from zero. (See tests C46012A..Z.)

The method used for rounding to longest integer is apparently round away from zero. (See tests C46012A..Z.)

The method used for rounding to integer in static universal real expressions is apparently round away from zero. (See test C4A014A.)

- Array types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT. For this implementation:

Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components are rejected at compile time when they are detected as containing dimensions with more than SYSTEM.MAX_INT components. (See test C36003A.)

NUMERIC_ERROR is raised when 'LENGTH is applied to an array type with INTEGER'LAST + 2 components. NUMERIC_ERROR is raised when an array type with INTEGER'LAST + 2 components is declared. (See test C36202A.)

NUMERIC_ERROR is raised when 'LENGTH is applied to an array type with SYSTEM.MAX_INT + 2 components. NUMERIC_ERROR is raised when an array type with SYSTEM.MAX_INT + 2 components is declared. (See test C36202B.)

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC ERROR when the array type is declared. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array subtype is declared. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Discriminated types.

During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications. (See test E38104A.)

In assigning record types with discriminants, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Aggregates.

In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, not all choices are evaluated before being checked for identical bounds. (See test E43212B.)

All choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

- Representation clauses.

The Ada Standard does not require an implementation to support representation clauses. If a representation clause is not supported, then the implementation must reject it.

Enumeration representation clauses containing noncontiguous values for enumeration types other than character and boolean types are supported. (See tests C35502I..J, C35502M..N, and A39005F.)

Enumeration representation clauses containing noncontiguous values for character types are supported. (See tests C35507I..J, C35507M..N, and C55B16A.)

Enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1) are supported. (See tests C35508I..J and C35508M..N.)

Length clauses with SIZE specifications for enumeration types are supported. (See test A39005B.)

Length clauses with STORAGE_SIZE specifications for access types are supported. (See tests A39005C and C87B62B.)

Length clauses with STORAGE_SIZE specifications for task types are supported. (See tests A39005D and C87B62D.)

Length clauses with SMALL specifications are supported. (See tests A39005E and C87B62C.)

Record representation clauses are not supported. (See test A39005G.)

Length clauses with SIZE specifications for derived integer types are supported. (See test C87B62A.)

- Pragmas.

The pragma INLINE is supported for procedures. The pragma INLINE is supported for functions. (See tests LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F.)

- Input/output.

The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

The package DIRECT_IO cannot be instantiated with unconstrained array types with discriminants without defaults. (See test EE2401D.)

Modes IN_FILE and OUT_FILE are supported for SEQUENTIAL_IO. (See tests CE2102D and CE2102E.)

Modes IN_FILE, OUT_FILE, and INOUT_FILE are supported for DIRECT_IO. (See tests CE2102F, CE2102I, and CE2102J.)

RESET and DELETE are supported for SEQUENTIAL_IO and DIRECT_IO. (See tests CE2102G and CE2102K.)

Dynamic creation and deletion of files are supported for SEQUENTIAL_IO and DIRECT_IO. (See tests CE2106A and CE2106B.)

Overwriting to a sequential file truncates the file to last element written. (See test CE2208B.)

An existing text file can be opened in OUT_FILE mode, cannot be created in OUT_FILE mode, and cannot be created in IN_FILE mode. (See test EE3102C.)

Only one internal file can be associated with each external file for text I/O for both reading and writing. (See tests CE2110B, CE2111D, CE3111B..E (4 tests), CE3114B, and CE3115A.)

More than one internal file can be associated with each external file for sequential I/O for both reading or both writing but not reading and writing. (See tests CE2107A..B (2 tests).)

Only one internal file can be associated with each external file for sequential I/O for both reading and writing. (See tests CE2107C..D (2 tests) and CE2111D.)

More than one internal file can be associated with each external file for direct I/O for both reading or both writing but not reading and writing. (See tests CE2107E..G (3 tests).)

Only one internal file can be associated with each external file for direct I/O for both reading and writing. (See tests CE2107H..I (2 tests) and CE2111H.)

An external file associated with more than one internal file cannot be deleted for SEQUENTIAL_IO, DIRECT_IO, and TEXT_IO. (See test CE2110B.)

Temporary sequential files are not given names. Temporary direct files are not given names. (See tests CE2108A and CE2108C.)

- Generics.

Generic subprogram declarations and bodies can not be compiled in separate compilations. (See tests CA1012A and CA2009F.)

Generic package declarations and bodies cannot be compiled in separate compilations. (See test CA2009C.)

Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

At the time of testing, version 1.9 of the ACVC comprised 3122 tests of which 25 had been withdrawn. Of the remaining tests, 241 were determined to be inapplicable to this implementation. Modifications to the code, processing, or grading for 39 tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	109	1047	1620	17	17	46	2856
Failed	0	0	0	0	0	0	0
Inapplicable	1	4	235	0	1	0	241
Withdrawn	3	2	19	0	1	0	25
TOTAL	113	1053	1874	17	19	46	3122

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	186	499	549	248	166	98	142	326	135	36	232	3	236	2856
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	18	74	126	0	0	0	1	1	2	0	2	0	17	241
Withdrawn	2	13	2	0	0	1	2	0	0	0	2	1	2	25
TOTAL	206	586	677	248	166	99	145	327	137	36	236	4	255	3122

3.4 WITHDRAWN TESTS

The following 25 tests were withdrawn from ACVC Version 1.9 at the time of this validation:

B28003A	E28005C	C34004A	C35502P	A35902C	C35904A
C35A03E	C35A03R	C37213H	C37213J	C37215C	C37215E
C37215G	C37215H	C38102C	C41402A	C45614C	A74106C
C85018B	C87B04B	CC1311B	BC3105A	AD1A01A	CE2401H
CE3208A					

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 241 tests were inapplicable for the reasons indicated:

C24113H..K (4 tests) contain literals which exceed the maximum line length for this implementation (120 characters).

C35702A uses SHORT_FLOAT which is not supported by this implementation.

C36003A is rejected at compile time when array type and subtype declarations are detected as containing dimensions with more than SYSTEM.MAX_INT components.

A39005G uses a record representation clause which is not supported by this compiler.

C45231D and B86001DT require a predefined numeric type other than those defined by the Ada language in package STANDARD. There is no such type for this implementation.

C45531M, C45531N, C45532M, and C45532N use fine 48 bit fixed point base types which are not supported by this compiler.

C45531O, C45531P, C45532O, and C45532P use coarse 48 bit fixed point base types which are not supported by this compiler.

C4A013B uses a static value that is outside the range of the most accurate floating point base type. The declaration was rejected at compile time.

C96005B requires the range of type DURATION to be different from those of its base type; in this implementation they are the same.

CA2009C compiles generic package body subunit specifications and bodies in separate compilations. This compiler requires that generic package subunit specifications and bodies be in a single compilation.

CA2009F compiles generic subprogram declarations and bodies in separate compilations. This compiler requires that generic subprogram declarations and bodies be in a single compilation.

BC3204C and BC3205D compile generic package specifications and bodies in separate compilations. This compiler requires that generic package bodies be in a single compilation.

CE2107C..D (2 tests), CE2107H..I (2 tests), CE2108A, CE2108C, CE3112A require temporary files be given names. This implementation does not give names to temporary files.

CE2110B, CE2111D, CE2111H, CE3111B..C (2 tests), CE3114B, and CE3115A are inapplicable because multiple internal files cannot be associated with the same external file with different file access modes. The proper exception is raised when multiple access is attempted.

EE2401D uses instantiations of package DIRECT_IO with unconstrained array types as the element type. These instantiations are rejected by this compiler.

CE3111D and CE3111E require that, when separate internal files output text to the same external file, the output from one internal file overwrite text output from the other file. The ONE PLUS Ada implementation does not use such overwrite semantics, but writes the values to the external file in sequence. The AVO ruled thar this behavior is acceptable, pending further review from the ARG.

The following 201 tests require a floating-point accuracy that exceeds the maximum of 15 digits supported by this implementation:

C24113L..Y (14 tests)	C35705L..Y (14 tests)
C35706L..Y (14 tests)	C35707L..Y (14 tests)
C35708L..Y (14 tests)	C35802L..Z (15 tests)
C45241L..Y (14 tests)	C45321L..Y (14 tests)
C45421L..Y (14 tests)	C45521L..Z (15 tests)
C45524L..Z (15 tests)	C45621L..Z (15 tests)
C45641L..Y (14 tests)	C46012L..Z (15 tests)

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made with the approval of the AVO, and are made in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into sub-tests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that wasn't anticipated by the test (such as raising one exception instead of another).

The following Class B tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B33301A	B55A01A	B67001A	B67001C	B67001D
B97103E	BC1109A	BC1109C	BC1109D	BC3205C

The following Class B tests were split because syntax errors at one point resulted in the compiler flagging correct code as being incorrect:

B22003A	B26001A	B26002A	B26005A	B28001D
B29001A	B37106A	B37301B	B38003A	B38003B
B38009A	B38009B	B54A01C	B61001C	B61001F
B61001H	B61001I	B61001M	B61001R	B61001W
B97101E	B97104G	BC1109A	BC1109C	BC1109D
BC1202B	BC1202E			

The following executable test was split because the resulting program was too large to be executed:

C35A06N

CE3602A attempts to create an external file with the same name (CE3602A) as the program file itself and in the same directory. For this implementation, this is an illegal file name duplication which raises a USE_ERROR exception unless the file name is changed or the program file name is located in a different directory from that of the created external file name. The AVF manager changed the executable file image name by the appending of the letter "X" to it, making it 'CE3602AX'. Both the executable file image name and the created external file name were located in the same directory and executed with no errors.

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.9 produced by the ONE PLUS Ada Compiler, Version 1.1 was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the ONE PLUS Ada Compiler, Version 1.1 using ACVC Version 1.9 was conducted on-site by a validation team from the AVF. The configuration consisted of a Host: DPS 6 PLUS/420 operating under HVS 6 PLUS V1.0 and a Target: DPS 6 PLUS/420 host operating under HVS 6 PLUS V1.0

A magnetic tape containing all tests except for withdrawn tests was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring modifications during the prevalidation testing were included in their modified form on the magnetic tape. The contents of the magnetic tape were loaded directly onto the host computer.

After the test files were loaded to disk, the full set of tests was compiled and linked on the DPS 6 PLUS/420, and all executable tests were run on the DPS 6 PLUS/420. Results were printed from the host computer.

The compiler was tested using command scripts provided by Honeywell Bull, Inc. and reviewed by the validation team. The compiler was tested using all default (option/switch) settings except for the following:

Option/Switch	Effect
-LIB libraryname	This required argument specifies the library file to be used for the compilation being invoked.

Tests were compiled, linked, and executed (as appropriate) using a single host computer and a single target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF.

3.7.3 Test Site

The validation team arrived at Billerica, MA on 11 January 1988, and departed after testing was completed on 15 January 1988.

APPENDIX A

CONFORMANCE STATEMENT

Appendix A

DECLARATION OF CONFORMANCE

Compiler Implementer: Honeywell Bull, Inc.
A.V.F.: Institute for Computer Sciences and Technology, NBS
Ada Compiler Validation Capability (ACVC) Version: 1.9

Base Configuration

Base Compiler Name:	ONE PLUS Ada Compiler	Version:	V1.1
Host Architecture	- ISA: DPS 6 PLUS/420	OS & VER#:	HVS 6 PLUS V1.0
Target Architecture	- ISA: DPS 6 PLUS/420	OS & VER#:	HVS 6 PLUS V1.0

Derived Compiler Registration

Derived Compiler Name:	ONE PLUS Ada Compiler	Version:	V1.1
Host Architecture	- ISA: DPS 6 PLUS/400	OS & VER#:	HVS 6 PLUS V1.0
Target Architecture	- ISA: DPS 6 PLUS/400	OS & VER#:	HVS 6 PLUS V1.0

Derived Compiler Name:	ONE PLUS Ada Compiler	Version:	V1.1
Host Architecture	- ISA: DPS 6 PLUS/410	OS & VER#:	HVS 6 PLUS V1.0
Target Architecture	- ISA: DPS 6 PLUS/410	OS & VER#:	HVS 6 PLUS V1.0

Implementer's Declaration

I, the undersigned, representing Honeywell Bull, Inc., have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compilers listed in this declaration. I declare that Honeywell Bull, Inc. is the owner of record of the Ada language compilers listed above and, as such, is responsible for maintaining said compilers in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compilers listed in this declaration shall be made only in the owner's corporate name.

Alan C. Lyman
Alan C. Lyman
Manager, Compiler Development

9/25/87
Date

Owner's Declaration

I, the undersigned, representing Honeywell Bull, Inc., take full responsibility for implementation and maintenance of the Ada compilers listed above, and agree to the public disclosure of the final Validation Summary Report. I further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. I declare that all of the Ada language compilers listed, and their host/target performance are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A. I have reviewed the Validation Summary Report for the compilers and concur with the contents.

Alan C. Lyman
Alan C. Lyman
Manager, Compiler Development

9/25/87
Date

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of MIL-STD-1815A, and to certain *allowed restrictions on representation clauses.* The implementation-dependent characteristics of the ONE PLUS Ada Compiler, Version 1.1, are described in the following sections which discuss topics in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A).. Implementation-specific portions of the package STANDARD are also included in this appendix.

```
package STANDARD is


    type INTEGER is range -32768 .. 32767;
    type SHORT_INTEGER is range -128 .. 127;
    type LONG_INTEGER is range  -2_147_483_648 .. 2_147_483_647;

    type FLOAT is digits 6 range -16#0.FFFFF8#E63 .. 16#0.FFFFF8#E64;
    type LONG_FLOAT is digits 15 range -16#0.FFFF_FFFF_FFFF_E0#E63
                                     .. 16#0.FFFF_FFFF_FFFF_E0#E63;

    type DURATION is delta 2**(-14) range -131_072.0 .. 131_071.0;


end STANDARD;
```

APPENDIX F OF THE Ada STANDARD

ATTACHMENT 4

IMPLEMENTATION-DEPENDENT CHARACTERISTICS

(per APPENDIX F of the Ada Standard)

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of MIL-STD-1815A, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of ONE PLUS Ada R4.1 and V1.1 are described in the following sections which discuss topics one through eight as stated in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A). Two other sections, package STANDARD and file naming conventions, are also included in this appendix.

(1) Implementation-Dependent Pragmas

None

(2) Implementation-Dependent Attributes

None

(3) Package SYSTEM

The specification for the package SYSTEM IS:

```
  package SYSTEM is

    type ADDRESS is ACCESS INTEGER;
    type NAME is (DPS6, DPS6_PLUS);

    SYSTEM_NAME    : constant NAME  := DPS6;
    STORAGE_UNIT   : constant := 16;
    MEMORY_SIZE    : constant := 2**20;

    --System-Dependent Named Numbers:

    MIN_INT        : constant := -2**31;
    MAX_INT        : constant := 16#7FFFFFFF#;
    MAX_DIGITS     : constant := 15;
    MAX_MANTISSA   : constant := 31;
    FINE_DELTA     : constant := 2.0**(-30);
    TICK           : constant := 1.0/120.0;

    -- Other System-Dependent Declarations

    subtype PRIORITY is INTEGER range 0 .. 16;

end SYSTEM;
```

IMPLEMENTATION-DEPENDENT CHARACTERISTICS (cont'd)

(4) Representation Clause Restrictions

In general, no representation clauses may be given for a derived type. The representation clauses that are accepted for non-derived types are described in the following:

Address Clause

Not supported

Length Clause

The compiler accepts only a length clause that specifies the number of storage units to be reserved for a collection.

Enumeration Representation Clause

Enumeration representation clauses may specify representations only in the range of predefined type INTEGER.

Record Representation Clause

A component clause is allowed if and only if:

- o The component type is a discrete type different from LONG_INTEGER
- o The component type is an array type with a discrete element type different from LONG_INTEGER.

No component clause is allowed if the component type is not covered by the above two inclusions. If the record type contains components not covered by a component clause, they are allocated consecutively after the component with the highest at value. Allocation of a record component without a component clause is always aligned on a storage unit boundary. Holes created because of component clauses are not otherwise utilized by the compiler.

(5) Conventions

No names denoting implementation-dependent components are generated.

(6) Address Clauses

Not supported

IMPLEMENTATION-DEPENDENT CHARACTERISTICS (cont'd)

(7) Unchecked Conversions

Unchecked conversion is only allowed between values of the same size. In this context the size of an array is equal to that of two access values and the size of a packed array is equal to two access values and an integer. This is the only restriction imposed on unchecked conversion.

(8) Input-Output Packages

SEQUENTIAL_IO Package

```
type FILE_TYPE is NEW BASIC_IO.FILE_TYPE;
procedure READ (FILE: in FILE_TYPE; ITEM: out ELEMENT_TYPE);
procedure WRITE (FILE: in FILE_TYPE; ITEM: in ELEMENT_TYPE);
function END_OF_FILE (FILE: in FILE_TYPE) return BOOLEAN;
```

DIRECT_IO Package

```
type COUNT is range 0 .. 16#7FFFFFFF#;
```

TEXT_IO Package

```
type COUNT is range 0 .. 16#7FFFFFFF#;

subtype FIELD is INTEGER range 0 .. 35;
```

LOW_LEVEL_IO

Low-level input-output is not provided

(9) Package STANDARD

```
type INTEGER is -32768 .. 32767;
type SHORT_INTEGER is -128 .. 127;
type LONG_INTEGER is -2_147_483_648 .. 2_147_483_647;

type FLOAT is digits 6 range -16#0.FFFFF8#E63 .. 16#0.FFFFF8#E63;
type SHORT_FLOAT is not defined
type LONG_FLOAT is digits 15 range -16#0.FFFF_FFFF_FFFF_E0#E63
                             .. 16#0.FFFF_FFFF_FFFF_E0#E63;

type DURATION is delta 2**(-14)
                    range -131_072.0 .. 131_071.0
```

(10) File Names

File names follow the conventions and restrictions of the target operating system.

APPENDIX C

TEST PARAMETERS

ATTACHMENT 3

PARAMETERS FOR ".TST" TESTS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are identified by names that begin with a dollar sign. A value is substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning	Value
$BIG IDI Identifier of size MAX_IN_LEN with varying last character.	AAA.......................AA1 (119 A'S followed by 1)
$BIG_ID2 Identifier of size MAX_IN_LEN with varying last character.	AAA.......................AA2 (119 A's followed by 2)
$BIG_ID3 Identifier of size MAX_IN_LEN with varying middle character.	AAA......... A3A.........AA (59 A's followed by 3, followed by 60 A's)
$BIG_ID4 Identifier of size MAX_IN_LEN with varying middle character.	AAA...........A4A.........AA (59 A's followed by 4, followed by 60 A's)
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is MAX_IN_LEN characters long.	000.......................00298 (117 0's followed by 298)

".TST" TEST PARAMETERS (cont'd)

Name and Meaning	VALUE
$BIG_REAL_LIT A real literal that can be either of floating- or fixed point type, has value 690.0, and has enough leading zeroes to be MAX_IN_LEN characters long.	000....................069.0E1 (114 0's followed by 69.0E1)
$BIG_STRING1 A string literal which is the same as the first half of $BIG_ID1.	"AAA.......................AA" (60 A's enclosed in quotes)
$BIG_STRING2 A string literal which is the same as the second half of $BIG_ID1.	"AAA.......................A1" (59 A's followed by 1 all enclosed in quotes)
$BLANKS Blanks of length MAX_IN_LEN - 20	100 blanks
$COUNT_LAST Value of COUNT'LAST in TEXT_IO package.	16#7FFFFFFF#
$FIELD_LAST Value of FIELD'LAST in TEXT_IO package.	35
$FILE_NAME_WITH_BAD_CHARS An illegal external file name that either contains invalid characters or is too long.	^BAD@CHAR[S]
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	XYZ*
$GREATER_THAN_DURATION A universal real value that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	131070.0
$GREATER_THAN_DURATION_BASE_LAST The universal real value that is greater than DURATION'BASE'LAST.	131071.5
$ILLEGAL_EXTERNAL_FILE_NAME1 Illegal external file name.	^BAD@CHAR[S]

ATTACHMENT 3

".TST" TEST PARAMETERS (cont'd)

Name and Meaning	Value
$ILLEGAL_EXTERNAL_FILE_NAME2 Illegal external file names.	TOO_LONG_A_FILE_NAME
$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST	-32768
$INTEGER_LAST The universal integer literal expression whose value is INTEGER'LAST.	32767
$INTEGER_LAST_PLUS_1 The universal_integer literal expression whose value is INTEGER'LAST + 1	32768
$LESS_THAN_DURATION A universal real value that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-131072.5
$LESS_THAN_DURATION_BASE_FIRST The universal real value that is less than DURATION'BASE'FIRST.	-131072.5
$MAX_DIGITS Maximum digits supported for floating-point types.	15
$MAX_IN_LEN Maximum input line length permitted by the implementation	120
$MAX_INT The universal_integer literal whose value is SYSTEM.MAX_INT	2147483647
$MAX_INT_PLUS_1 The universal_integer literal whose value is SYSTEM.MAX_INT+1	2147483648
$MAX_LEN_INT_BASED_LITERAL A universal ~~real~~ INTEGER based literal (using colons) whose value is 2:11:, but with enough leading zeroes in the mantissa so that the literal is MAX_IN_LEN characters long.	2:00...................011: (115 0's)

".TST TEST PARAMETERS (cont'd)

Name and Meaning	Value
$MAX_LEN_REAL_BASED_LITERAL A universal_real based literal (using colons) whose value is 16:F.E:, but with enough leading zeroes in the mantissa so that the literal is MAX_IN_LEN characters long.	16:00....................0F.E: (113 0's)
$MAX_STRING_LITERAL A string literal that is MAX_IN_LEN characters long (including the quote characters).	"AAA.......................AA" (118 A's all enclosed in quotes)
$MIN_INT The (signed) universal_integer literal whose value is SYSTEM.MIN_INT.	-2147483648
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	none
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFF#

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 25 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

B28003A	A basic declaration (line 36) wrongly follows a later declaration.
E28005C	This test requires that 'PRAGMA LIST (ON);' not appear in a listing that has been suspended by a previous "pragma LIST (OFF);"; the Ada Standard is not clear on this point, and the matter will be reviewed by the ALMP.
C34004A	The expression in line 168 wrongly yield a value outside of the range of the target type T, raising CONSTRAINT_ERROR.
C35502P	The equality operators in lines 62 and 69 should be inequality operators
A35902C	Line 17's assignment of the nominal upper bound of a fixed-point type to an object of that type raises CONSTRAINT_ERROR, for that value lies outside of the actual range of the type.
C35904A	The elaboration of the fixed-point subtype on line 28 wrongly raises CONSTRAINT_ERROR, because its upper bound exceeds that of the type.
C35A03E & R	These tests assume that attribute 'MANTISSA returns 0 when applied to a fixed-point type with a null range, but the Ada Standard does not support this assumption.
C37213H	The subtype declaration of SCONS in line 100 is wrongly expected to raise an exception when elaborated.
C37213J	The aggregate in line 451 wrongly raises CONSTRAINT_ERROR.
C37215C, E,G,H	Various discriminant constraints are wrongly expected to be incompatible with the type CONS.
C38102C	The fixed-point conversion on line 23 wrongly raises CONSTRAINT_ERROR.

C41402A	'STORAGE_SIZE is wrongly applied to an object of an access type.
C45614C	REPORT.IDENT_INT has an argument of the wrong type (LONG_INTEGER).
A74106C C85018B C87B04B CC1311B	A bound specified in a fixed-point subtype declaration lies outside of that calculated for the base type, raising CONSTRAINT_ERROR. Errors of this sort occur re lines 37 & 59, 142 & 143, 16 & 48, and 252 & 253 of the four tests, respectively (and possibly elsewhere)
BC3105A	Lines 159..168 are wrongly expected to be incorrect; they are correct.
AD1A01A	The declaration of subtype INT3 raises CONSTRAINT_ERROR for implementations that select INT'SIZE to be 16 or greater.
CE2401H	The record aggregates in lines 105 and 117 contain the wrong values.
CE3208A	This test expects that an attempt to open the default output file (after it was closed) with MODE_IN file raises NAME_ERROR or USE_ERROR; by commentary AI-00048, MODE_ERROR should be raised.